Deep Graph Node Kernels: a Convex Approach

Luca Oneto, *Member, IEEE*, Nicolò Navarin, *Member, IEEE*,
Alessandro Sperduti, *Senior Member, IEEE*, and Davide Anguita, *Senior Member, IEEE*

***Abstract*— Nowadays, developing effective techniques able to deal with data coming from structured domains is becoming crucial. In this context kernel methods are the state-of-the-art tool widely adopted in real-world applications that involve learning on structured data. Contrarily, when one has to deal with unstructured domains, deep learning methods represent a competitive, or even better, choice. In this paper we propose a new family of kernels for graphs which exploits a deep representation of the information. Our proposal exploits the advantages of the two worlds. From one side we exploit the potentiality of the state-of-the-art graph kernels. From the other side we develop a deep architecture through a series of stacked kernel pre-image estimators trained in an unsupervised fashion via convex optimization. The hidden layers of the proposed framework are trained in a forward manner and this allows us to avoid the greedy layerwise training of classical deep learning. Results on real world graph datasets confirm the quality of the proposal.**

I. Introduction

KERNEL methods (KM) are powerful learning algorithms that can easily be applied to every input domain [1] and are backed up by Statistical Learning Theory (SLT) that offers strong theoretical guarantees on the output hypothesis [2]. These methods represent the output hypothesis in terms of pairwise similarity among the examples and do not work on an explicit representation of them [3]. The function used to compute the similarity must be a kernel function. KM consistently outperformed previous generations of learning techniques [4], [1], [5], [6]. They provide a flexible and expressive learning framework that has been successfully applied to a wide range of real world problems but, recently, novel algorithms, such as Deep Neural Networks and Ensemble Methods [7], [8], have increased their competitiveness against them. In particular, when one has to deal with data coming from the unstructured domain, Deep Learning (DL) techniques are quite a powerful tool for extracting an informative representation of the data [9], [10], [11], [12], [13].

Due to the current data growth in size, heterogeneity and structure, the new generation of algorithms are expected to solve increasingly challenging problems.

However, there are several domains where it is not straightforward to define a vectorial representation for the data, but it is easy to encode it in a structured form. For example, recommendation systems can be seen as link-prediction systems in bipartite graphs where nodes represent users and items [14]. Moreover, social networks can be represented as a graph where nodes are the users and links are their interactions; in this setting, it is interesting to predict (i.e. suggest to users) possible novel connections [15]. Other examples include citations between scientific articles, or networks in linguistics [16], In these and other scenarios, the learning problem can be expressed as classification (or ranking) over the nodes in a single, big graph. In this context, KM constitute a well established machine learning approach for structured domains because they can be defined directly on structured data [17]. This relieves the user from the definition of a vectorial representation of the data, a time consuming and task-specific operation. Given an input domain, e.g. a graph domain, the user is required to fix the specific kernel function and the corresponding parameters to adopt. Given a kernel function, the problem of reconstructing the input patterns from the mapping in the feature space induced by the kernel is usually referred to as pre-image estimation [18]. Different kernel functions for graphs are available in literature, each one encoding a (slightly) different similarity measure between graph nodes. In general, the node similarity is defined in terms of their distance, in the sense of how many steps have to be performed to go from one node to another one in the graph. Usually, these kernels map the graph nodes in an infinite-dimensional feature space. Thus, the explicit feature map of the kernels cannot be computed. However, the kernels can be efficiently computed in a closed form by matrix operations on the adjacency matrix of the graph.

On the other side, when one has to deal with data coming from an unstructured domain, DL techniques represent a competitive, or even better, choice. In fact, before learning, feature learning is required and conducted in many applications in order to achieve a satisfactory accuracy [19], [20], [21]. DL represents a state-of-the-art choice in this context [10], [11], [12], [13]. The deep architecture extracts features by a multilayer feature representation framework, and the higher layers represent more abstract information than those from the lower ones. Standard DL considers multilayer as a whole with unsupervised initialization, and after such initialization the entire network is trained by back propagation and all of the layers are fine tuned together [12]. Note that all the hidden parameters in DL framework need to be fine-tuned multiple times and are affected by the problem of local minima and slow convergence rate. This results in a cumbersome and time consuming procedure [22], [9]. Recently some attempts have been made to overcome these limitations. The most successful proposal is the one of

Luca Oneto and Davide Anguita are with the DIBRIS Department, University of Genoa, Via Opera Pia 13, I-16145, Genova, Italy (email: {Luca.Oneto, Davide.Anguita}@unige.it). Nicolò Navarin and Alessandro Sperduti are with the Department of Mathematics, University of Padova, Via Trieste, 63, I-35121 Padova - Italy (email: {nnavarin,sperduti}@math.unipd.it). This work was partly supported by the University of Padova under the strategic project BIOINFOGEN.

[9] which develops a deep architecture which exploits the principle behind the Extreme Learning Machines random feature mapping together with a series of stacked sparse autoencoders which do not require fine-tuning.

In the field of learning on graphs, few work has been done in the direction of creating deep graph kernels. [23] proposes a method to learn the similarity between features in the feature space in the case of kernels for graphs (i.e. kernel functions defined between two different graphs), using language modeling and deep learning techniques. In this case, an explicit representation for the features is required. In the case of graph node kernels, such an explicit representation is not straightforward to define, and existing graph node kernels are defined only implicitly (i.e. they do not explicitly compute the features). [24] learns (online) a compact representation for graph nodes. This technique uses deep learning as a tool to build node representations, but it analyzes only a subsampling of short random walks (that have to be stored explicitly), thus the expressivity of the resulting representation is limited.

In this paper we propose a new family of Kernels for Graph nodes (GK) which exploits the advantages of the two worlds. From one side we exploit the potentiality of the state-of-the-art kernels for graph nodes which are able to give an implicit powerful and expressive representation. From the other side we develop a deep architecture through a series of stacked kernel pre-image estimators trained in an unsupervised fashion via convex optimization. Our approach is inspired by the works of [22], [9] but, instead of building the hidden representation via random projection, the state-of-the-art GK are employed. The hidden layers of the proposed framework are trained in a forward manner and, thanks to this approach, the final architecture is the result of a series of convex problems which do not suffer from problems of local minima and slow rate of convergence. This allows us to avoid the greedy layerwise training of classical DL. We will show how the proposed Deep Graph Kernels (DGK) perform with respect to the state-of-the-art ones on a series of real world graph datasets. Results confirm the quality of the proposal.

The paper is organized as follows. Section II recalls the learning framework. Section III recalls the state-of-the-art GK. In Section IV we describe our proposal of a convex approach for generating a new family of DGK. In Section V we compare our proposal with the state-of-the-art on a series of real world graph datasets. Section VI concludes the paper.

II. Learning with Kernels

In this paper, we will deal with multiclass classification problems [2]: based on some random observations X of the input space $\mathcal{X}$, one has to estimate the associated label Y which belongs to the output space composed of c classes $\mathcal{Y} = \{1, 2, \cdots, c\}$ by choosing a suitable hypothesis $h : \mathcal{X} \to \mathcal{Y}$, in a set of possible ones $\mathcal{H}$. A learning algorithm selects $h \in \mathcal{H}$ by exploiting a set of labeled samples $\mathcal{D}_n : \{(X_1, Y_1), \cdots, (X_n, Y_n)\}$. The latter are sampled i.i.d. according to the distribution μ over the cartesian product between the input and output space $\mathcal{X} \times \mathcal{Y}$. The generalization error

$$L(h) = \mathbb{E}_{(X,Y)} \ell(h(X), Y), \tag{1}$$

associated to an hypothesis $h \in \mathcal{H}$, is defined through a loss function $\ell(h(X), Y) : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$. As μ is unknown, $L(h)$ cannot be explicitly computed, thus we have to resort to its empirical estimator, namely the empirical error

$$\widehat{L}_n(h) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(h\left(X_i\right), Y_i\right). \tag{2}$$

Let us begin by considering the binary classification problems where $\mathcal{Y} = \{\pm 1\}$. A simple criterium for selecting the final model during the training phase consists in choosing the approximating function that minimizes the empirical error $\widehat{L}_n(h)$: this approach is known as Empirical Risk Minimization (ERM) [2]. However, ERM is usually avoided as it leads to severely overfitting the model on the training dataset [2]. A more effective approach consists in the minimisation of a cost function where the trade-off between accuracy on the training data and a measure of the complexity of the selected approximating function is implemented [25]:

$$h^* : \quad \min_{h \in \mathcal{H}} \ \mathcal{C}(h) + C \widehat{L}_n(h), \tag{3}$$

where $\mathcal{C}(h)$ is a complexity measure which depends on the selected ML approach and $C \in [0, \infty)$ is a hyperparameter that must be set a priori and regulates the trade-off between the overfitting tendency, related to the minimization of the empirical error, and the underfitting tendency, related to the minimization of $\mathcal{C}(h)$. This approach is known as Structural Risk Minimization (SRM) [2].

Support Vector Machines (SVMs) represent one of the state-of-the-art algorithms for binary classification problems [2], [4]. In SVM, approximation functions are defined as

$$h(X) = \boldsymbol{w}^T \boldsymbol{\phi}(X) + b. \tag{4}$$

Moreover, in SVMs the loss function exploited is the hinge loss function $\ell\left(h\left(X_i\right), Y_i\right) = \max[0, 1 - Y_i h\left(X_i\right)]$ and the complexity measure is $\mathcal{C}(h) = \|\boldsymbol{w}\|^2$. The result of these choices is the primal formulation of SVM:

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T \boldsymbol{\phi}(X_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned} \tag{5}$$

By computing the dual formulation of Problem (5) it is possible to derive the following convex constrained quadratic programming problem (CCQPP):

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j Y_i Y_j \kappa(X_i, X_j) - \sum_{i=1}^{n} \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C, \end{aligned} \tag{6}$$

where $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel function which is a symmetric positive semi-definite function that corresponds

to a dot product in a Reproducing Kernel Hilbert Space (RKHS), i.e. there exists a $\phi : \mathcal{X} \rightarrow \mathcal{K} \subseteq \mathbb{R}^D$, where $\mathcal{K}$ is an Hilbert space (commonly referred to as *feature space*), such that $\kappa(X_i, X_j) = \langle \phi(X_i), \phi(X_j) \rangle$ with $X_i, X_j \in \mathcal{X}$ [26], [1]. Note that the input space $\mathcal{X}$ can be any space and that we do not need to know explicitly ϕ in order to solve Problem (6) and also to compute $h(X)$ since:

$$h(X) = \sum_{i=1}^{n} \alpha_i Y_i \kappa(X_i, X) + b. \tag{7}$$

The kernel κ usually is characterized by one or more hyperparameters. With $\gamma \in \Gamma$ we will indicate a combination of the hyperparameters in the set of all the possible ones.

The problem of SVM it that is only able to deal with binary classification problems. In order to extend SVM to multiclass problems in this paper we will exploit the All Versus All (AVA) approach [27], [28], [29]. In AVA the multiclass problem is split in all the possible binary classification problems which are $\binom{c}{2} = \frac{c(c-1)}{2}$ and solved via SVM. Then, every time a new sample X has to be classified each of the $\binom{c}{2}$ are applied and then label is assigned according to the majority voting criteria.

Finally, in SVM there is a need for tuning the hyperparameters which influence the algorithm performances. This procedure is called Model Selection (MS). The MS for SVM aims at tuning the following hyperparameters: the regularization hyperparameter C, the kernel κ and the hyperparameters of the kernels γ [30]. Moreover we need to Estimate the Error (EE) of the final classifier built with the best configuration of the hyperparameters [30]. A state-of-the-art technique for tuning the hyperparameters and estimating the error of the final classifier is the Nested K-Fold Cross Validation (KCV) [30], [31]. The nested KCV technique consists in splitting a dataset in K independent subsets and using, in turn, one set for EE, and the others to train a classifier after performing another *inner* KCV for MS on just the training data. Since the data in the different sets are i.i.d. the result of the procedure is unbiased.

Finally note that, when the input space $\mathcal{X}$ are the nodes in a graph, the main problem involves the development of kernels able to deal with such input domain (i.e. that are expressive enough fopr the considered task), and then choose one of these kernels, together with its values for the hyperparameters. In this setting, the challenge is to capture in the kernel the local and global structure of the graph. In the next section, we will present some of the state-of-the-art approaches.

III. Shallow Graph Kernels

Let $G = (V, E)$ be a graph, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices ($|V| = n$) and $E = \{(v, u) | u, v \in V\}$ is the set of edges ($|E| = m$). Every edge can have an associated weight $w_(u, v)$, that is zero by convention if $(u, v) \notin E$. Note that, in our setting, $\mathcal{X}$ is the space of nodes in the considered graph, and the labels we aim to predict are associated to the nodes in the graph. Given a graph G, let us define its adjacency matrix $\mathbf{A}_G$ with entries $\{\mathbf{A}_G\}_{ij} = w_{ij}$. From now on, when clear from the context, we will omit the subscript G. Let us define the laplacian matrix as $\mathbf{L}_G = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is a diagonal matrix with the entries set to the summation over the corresponding row of the adjacency matrix, i.e. $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. In the following, we will give the definition of four state-of-the-art kernels for graph nodes, that we will consider as baselines in the experimental section. All the kernels are based on the diffusion (or heat) principle, that is based on the heat equation that models the diffusion of heat in a system. Diffusion kernels on graphs are motivated by several interpretations [32]. These kernels are relatively fast to compute, and show good predictive performance [14], [32]. We will provide the kernel formulation on the whole Gram matrix, of size $n \times n$, starting from the graph G.

Laplacian Exponential Diffusion kernel (LEDK) [33]. It is closely related to continuous-time Markov chains [34]. Intuitively, let us define a quantity $x_i(t)$ associated to the vertex i at time t. This quantity diffuses to neighboring nodes with a symmetric diffusion rate A_{ij}. During a small time interval δt an amount $A_{ij} x_i \delta t$ is transferred from vertex i to vertex j. The balance equation is then

$$x_i(t + \delta t) = x_i(t) + \sum_{j=1}^{n} A_{ji} x_j \delta t - \sum_{j=1}^{n} a_{ij} x_j \delta t. \tag{8}$$

The kernel can be derived as

$$LEDK(G) = e^{-\beta \mathbf{L_G}}, \tag{9}$$

where $0 \leq \beta \leq 1$ is a parameter used to control the rate of diffusion.

Markov Diffusion kernel (MDK) [14], is based on the diffusion distance defined in [35], where the idea is that two nodes are considered similar if they diffuse in a similar way through the graph. It depends on a parameter t controlling the length of the considered walks. Let us define the probability transition matrix $\mathbf{P}$ as $\{\mathbf{P}\}_{ij} = \mathbf{A}_{ij}/\mathbf{D}_{ii}$. Moreover, let us define $\mathbf{Z}(t) = \frac{1}{t} \sum_{\tau=1}^{t} \mathbf{P}^{\tau}$. Then we can define the kernel as:

$$MDK(G) = \mathbf{Z}_G(t) \mathbf{Z}_G^T(t). \tag{10}$$

Markov Exponential Diffusion kernel (MEDK) [36]. This kernel tries to balance the similarity of two vertices regardless from the degree of the nodes (for other kernels, the similarity between two high-degree nodes is higher than the similarity between low-degree vertices). Let us define $\mathbf{M} = (\mathbf{D} - \mathbf{A} - n\mathbf{I})/n$ where n is the number of vertices in the graph. Then, MEDK is defined as

$$MEDK(G) = e^{-\beta \mathbf{M}}. \tag{11}$$

Regularized Laplacian Kernel (RLK) [33], is also referred to as the normalized random walk with restart kernel in [37]. It is defined as

$$RLK(G) = \sum_{n=1}^{\infty} \beta^n (-\mathbf{L})^n = (\mathbf{I} + \alpha \mathbf{L})^{-1}, \tag{12}$$

where $\alpha > 0$ is a parameter.

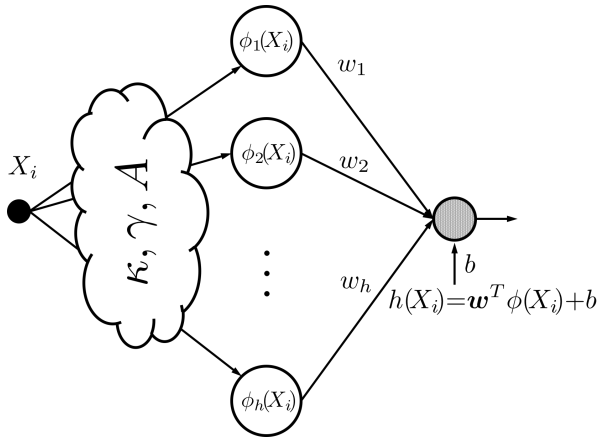


Fig. 1. The shallow architecture of the SVMs with a GK.

The kernels LEDK and MEDK require to compute a *matrix exponential*, that (e.g. using Sylvester's equations) has a complexity of $O(n^3)$. The kernel MDK computes t matrix multiplications, each one with a cost of approximately $O(n^3)$ (the fastest algorithm runs in $O(n^{2.373})$). The kernel RLK requires a matrix inversion (same complexity as matrix multiplication).

IV. Deep Graph Kernels

As described in the previous section, a graph G of n vertices $X_i \in V$ can be exhaustively represented by the adjacency matrix $A \in \mathbb{R}^{n \times n}$. In this graph just l of the n vertices are labelled with $Y \in \mathcal{Y}$. Our goal is to build a model which is able to predict the label of all the $n-l$ remaining vertices.

If we use the multiclass SVMs described in Section II together with the GK of Section III the result is equivalent to a shallow architecture. In particular, given a GK κ together with the values of its hyperparameters which maps a vertex $X_i \in V$ of the graph G into an unknown vector $\phi(X_i) \in \mathbb{R}^h$ based on the adjacency matrix A, and the labelled vertices identified with the set of indexes $\mathcal{I}^l$, it is possible to train a series of binary classifiers with the SVMs in order to obtain a multiclass classifier. Each of these SVMs binary classifiers is the result of a CCQPP whose aim is to find $\boldsymbol{w}$ and b in order to compute $h(X) = \boldsymbol{w}^T\phi(X)+b$. Since, ϕ is unknown we have to resort to its implicit formulation with the use of κ, $h(X) = \sum_{i \in \mathcal{I}^n} \kappa(X_i, X) + b$. Remember that MS and EE phases are needed in order to tune the hyperparameters and to estimate the error of the final classifier. As it is possible to see from Figure 1, this approach is equivalent to a neural network with a shallow architecture where the hidden nodes are not trained but implicitly defined a priori based on the chosen kernel and hyperparameters of the kernel. This shallow architecture has the advantage that it can be easily trained through a CCQPP. Fixed a kernel function, the pre-image problem is defined as follows: given a point $\Phi \in \mathcal{K}$ (the RKHS), find a corresponding pattern $x \in \mathcal{X}$ such that $\Phi = \phi(x)$. Since $\mathcal{K}$ is usually a far larger space than $\mathcal{X}$ (i.e. the mapping is typically not bijective), pre-image estimation is inherently illposed [18]. In these cases, an approximated pre-image z can be found as the solution of a minimization problem with constraints on the solution [38] (see eq. 14). This pre-image approximator has a de-nosing effect on the input data. In order to make the GK of Section III deep we will stack a series of pre-image estimators built through a particular procedure. In particular, given a GK κ instead of learning directly the classifiers, we try to learn the pre-images of the vertices (projected in the feature space), and stack a series of these estimators in order to build a deep architecture. The problem is that a vertex X_i, by definition, is not represented by a simple vector. For this reason we make an approximation by stating that a vertex X_i is represented by the column (or the row) ith of the adjacency matrix A. Basically the purpose of the pre-image estimator is to learn the adjacency matrix A (see Figure 2). The problem is that A contains real values and for this reason the SVMs cannot be applied since the problem becomes a regression problem. Consequently, we will exploit the approach of Eq. (3) but instead of using the hinge loss function we will use the square loss function while, as complexity term, we will still use the same as SVMs. In particular we have to learn a pre-image estimator able to map the ith column of A, that we will indicate with A_i, which represents X_i, into the same vector, and we have n available samples for this mapping, one for each vertex of G. Consequently we have n standard regression problems, one for each element $j \in \{1, \cdots, n\}$ of A_i, that we will call $A_{i,j}$. The model that we will use is a liner separator in the space induced by κ from X which is ϕ:

$$h_j(X) = \boldsymbol{w}_j^T\phi(X),\ j \in \{1, \cdots, n\}. \tag{13}$$

By adopting the approach of Eq. (3) with a square loss function and the squared norm of $\boldsymbol{w}_j$ as regularizers we obtain that:

$$\boldsymbol{w}_j^* = \arg\min_{\boldsymbol{w}_j} \sum_{i=1}^{v} (\boldsymbol{w}_j^T\phi(X_i) - A_{i,j})^2 + \lambda\|\boldsymbol{w}_j\|^2 \tag{14}$$

where λ is an hyperparameter which balances accuracy and complexity and must be tuned during the MS phase. By exploiting the representer theorem [26] and by simple linear algebra it is possible to obtain that:

$$h_j(X) = \sum_{i=1}^{v} \alpha_{i,j}\kappa(X_i, X), \tag{15}$$

and that $\boldsymbol{\alpha}_j$ can be retrieved as follows

$$\boldsymbol{\alpha}_j = (Q + \lambda I)^+ A_j, \tag{16}$$

where in the matrix Q the element $Q_{i,j} = \kappa(X_i, X_j)$, I is the identity matrix, $(\cdot)^+$ is the pseudoinverse of a matrix. By stacking many of these pre-image estimators, with different kernels or different kernel hyperparameters we obtain the DGK of Figure 3. Note that the pre-image estimators can be trained without the knowledge of the labels and consequently we can train the architecture over the whole adjacency matrix which contains also the unlabelled vertices. Moreover, note

that finding the weights of the whole DGK architecture is a convex problem which has an unique solution.

The final classifier can be built by using this new DGK inside the multiclass SVMs described in Section II.

V. Experimental Results

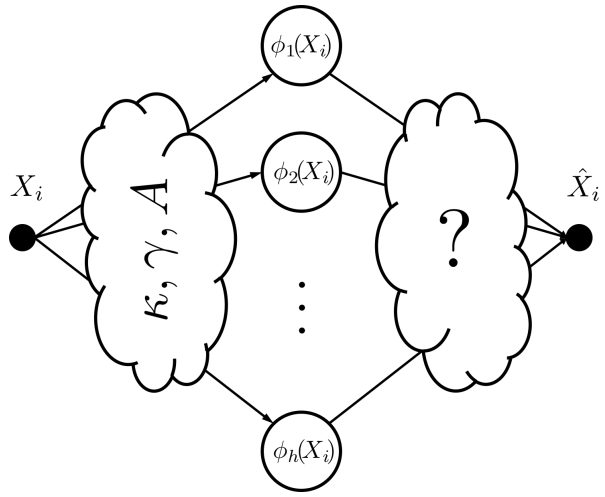
In this section we will show how the deep graph kernels proposed in Section IV perform with respect to the state-of-the-art ones of Section III. We will make use of a series of real word graph node classification problems that can be retrived from [39]. In particular we will focus on five problems: IMDB, IMDB_ALL, INDUSTRY_YH, INDUSTRY_PR, and WEBKB. A detailed description of each dataset is provided in the following.

- The IMDB dataset is based on data retrieved from the Internet Movie Database. This data was built to re-create an earlier study to build models predicting movie success as determined by box-office receipts. This data contains movies released in the United States between 1996 and 2001, with class labels identifying whether the opening weekend box-office receipts will exceed two million dollars. Two movies share a link if they share a production company. The weight of an edge in the resulting graph is the number of production companies two movies have in common.
- The IMDB_ALL dataset is similar to IMDB, but there is an edge connecting two movies if they share a production company, producer, director, or actor. The weight of an edge is the number of such entities two movies have in common.
- The INDUSTRY_YH dataset contains companies that are linked via cooccurrence in text documents. 22170 business news stories from the web have been collected between the 4th of January 1999 and the 8th of April 1999. An edge was created between two companies if they appeared together in a story. The weight of an edge is the number of such cooccurrences found in the complete corpus. The resulting network comprises 1798 companies that cooccurred with at least one other company. The labels of the companies are based on Yahoo!'s 12 industry sectors.
- The INDUSTRY_PR dataset is similar to INDUSTRY_YH, but based on 35318 PR Newswire press releases gathered from the 1st of April 2003 through the 30th of September 2003. The companies mentioned in each press release were extracted and an edge was placed between two companies if they appeared together in a press release. The weight of an edge is the number of such cooccurrences found in the complete corpus. The resulting network comprises 2189 companies that cooccurred with at least one other company.
- The WEBKB dataset is based on the WebKB Project [40]. It consists of sets of web pages from four computer science departments (Cornell, Wisconsin, Washington, Texas), with each page manually labeled into 7 categories: course, department, faculty, project, staff, student, or other. We do not include the 'other' pages in the graph, but use them to generate edges. This data file contains eight different graphs (two per university). For each university, we have the graph using direct hyperlinks and another graph using co-citation links (if x links to z and y links to z, then x and y are co-citing z). To create co-citation edges, we do allow an 'other' page as an intermediary although the final graph does not include the 'other' pages. To weight the link between x and y, we sum the number of hyperlinks from x to z and separately the number from y to z, and multiply these two quantities.

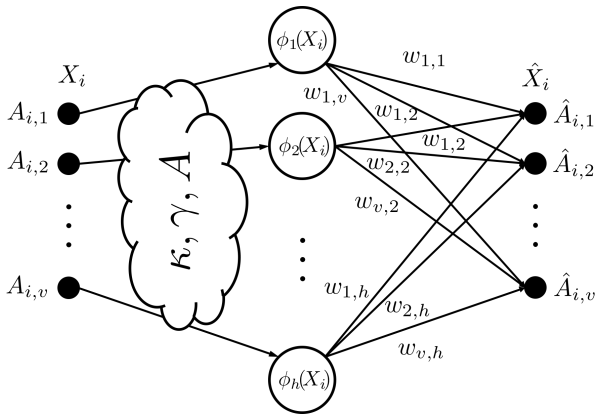
In Table I the average accuracy results of the considered base kernels, together with their deep version (applying the procedure detailed in Section IV) are reported. The first two columns refer to the LEDK kernel and its deep version (using the method proposed in this paper). The third and fourth columns refer to the MDK kernel and its deep version, while the fifth and sixth columns refer to the RLK kernel and its deep version, respectively. We reported in bold, for each dataset, the kernel with the best performance between the shallow and the deep one. We computed the average accuracy in 10-fold cross validation, where 9 folds are used as the test set, and just one fold is used for training a nested 5-fold cross validation where the hyper-parameters are selected. The whole experiment was repeated 10 times (with different splits for the CV), and the average accuracies are reported in the table. Note that we used just 10% of labels as training set. We also conducted experiments using 33% of the labels as training (i.e. 3-fold CV with 1 fold for training and 2 for testing), and obtained very similar results (not reported). The kernel parameters have been optimized in: $\beta \in \{0.01, \ldots, 0.05, 0.1, 0.3, 0.5\}$ for LEDK and MEDK; $\alpha \in \{10^{-2}, \ldots, 10^{3}\}$ for RLK; $t \in \{1, 2, 3, 5, 10, 50, 100\}$ for MDK. The λ parameter of the pre-image estimation layers (see eq. 14 and eq. 16) has been selected in $\{10^{-6}, .., 10^{4}\}$. The SVM C parameter (for the last layer) has been validated in the set $\{10^{-4}, .., 10^{4}\}$. We decided not to report the results of MEDK kernel because it performed poorly on all the considered datasets.

From the results in Table I emerges that, for the great majority of dataset/kernel combinations, the proposed approach is able to improve (in some cases by a large margin) the predictive performances of the base kernel. In particular, for LEDK kernel the performance improves in seven out of eight datasets, while for MDK in four out of eight and for RLK in three out of eight. Not that, when the performance of the deep kernel do not improve with respect to the relative base kernel, they are always comparable. Note also that, for each dataset, the best performing kernel is a deep one, confirming that not only our proposed method is able to boost the performance of the selected base kernel for the majority of the kernel/dataset combinations, but also that our approach is able to reach state-of-the-art performances in all the considered datasets.

Finally, it is worth to point out that, for the WEBKB_CORNELL, WEBKB_WISCONSIN and WE-



(a) Pre-image estimator for the vertex X_i of the graph G



(b) Pre-image estimator for the vertex X_i of the graph G described in term of the adjacency matrix with A_i

Fig. 2. The pre-image estimator of a DGK.

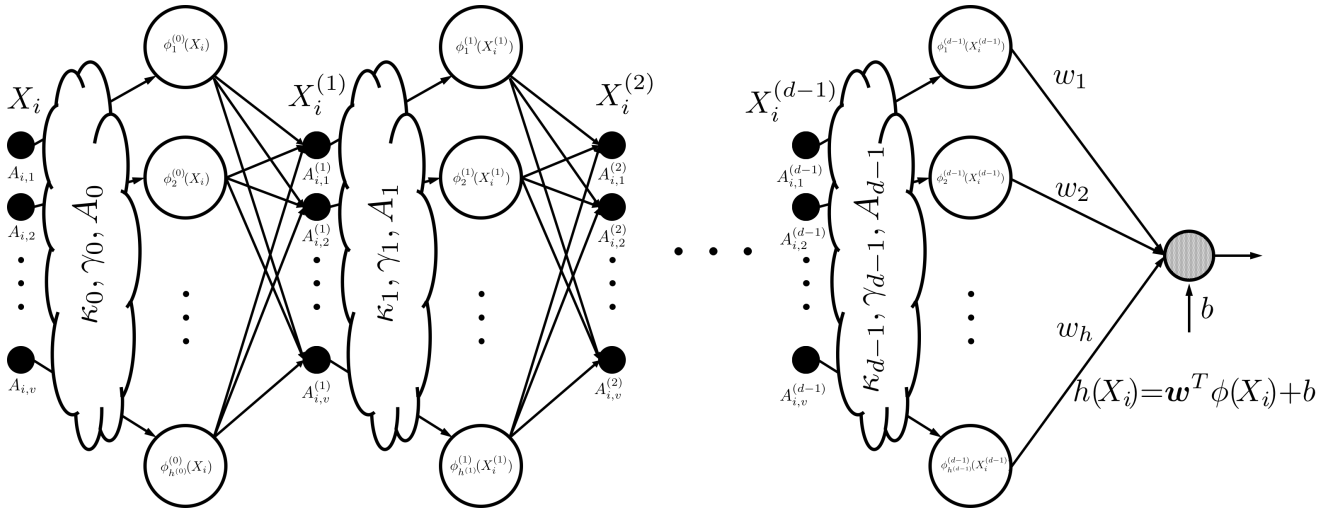


Fig. 3. The final deep architecture of the DGK.

BKB_TEXAS datasets, the RLK kernel does not encode a meaningful metric: indeed, the results show that the kernel behaves like an identity matrix, i.e. the predicted class is basically random. In these cases, the proposed approach is obviously not able to improve the performances of the base kernel.

VI. Conclusions

Developing techniques able to deal with data coming from structured domains is becoming every day more important. In this context kernel methods represent a state-of-the-art technique. In this paper we proposed a new family of kernels for graphs which exploits a deep representation of the information. Deep learning methods represented a breakthrough in learning unstructured domains because of their ability of giving a better representation of the input domain. Our proposal exploited the advantages of the two worlds. From one side we exploited the potentiality of the state-of-the-art graph kernels. From the other side we developed a deep architecture through a series of stacked kernel pre-image estimators trained in an unsupervised fashion via convex optimization. The hidden layers of the proposed framework were trained in a forward manner and this allowed us to avoid the greedy layerwise training of classical deep learning. Results on real world graph datasets confirmed the quality of the proposal.

Further investigation is needed in order to test the potentiality of our proposal. For example, we want to analyze the effect of the introduction of a first linear layer, performing a Singular Value Decomposition of the input, that would result in a more compact representation of the inputs for the deeper layers. Moreover, we need to better investigate the effect of the number of layers on the final performance. We also need to check if more complex combinations of graph kernels

Dataset/kernel	LEDK	DLEDK*	MDK	DMDK*	RLK	DRLK*
IMDB	73.05±2.39	**76.25**±1.23	**78.19**± 0.44	78.08±0.46	66.58± 1.50	**69.39**±1.44
IMDB_ALL	58.52±1.17	**75.42**±0.96	72.24± 0.45	**72.83**±0.55	72.36±2.17	**73.32**±2.06
INDUSTRY_PR	**34.28**±0.41	34.02±0.42	**36.72**±0.34	36.61±0.49	**34.53**±0.40	34.40±0.35
INDUSTRY_YH	31.99±0.38	**32.89**±0.53	**46.89**±0.42	46.32±0.36	30.51±0.41	**30.85**±0.73
WEBKB_CORNELL	42.86±0.42	**44.52**±0.84	53.99±1.02	**54.52**±0.73	41.31±0.00	41.31±0.00
WEBKB_WISCONSIN	53.42±0.95	**56.09**±1.22	70.34±1.10	**70.72**±0.99	43.78±0.00	43.78±0.00
WEBKB_WASHINGTON	46.07 ±1.36	**51.39**±2.50	**65.87**±0.70	65.39±0.64	**42.51**±1.74	39.34±0.85
WEBKB_TEXAS	58.08±0.97	**58.83**±0.74	68.11±0.82	**68.58**±0.75	48.22±0.00	48.22±0.00

TABLE I

EXPERIMENTAL RESULTS COMPARING THE PROPOSED APPROACH APPLIED TO 3 STATE-OF-THE-ART GRAPH NODE KERNELS ON 8 REAL-WORLD DATASETS, WITH 10% OF THE LABELS USED AS TRAINING SET. LEDK:LAPLACIAN EXPONENTIAL DIFFUZION KERNEL, MDK: MARKOV DIFFUSION KERNEL, RLK: REGULARIZED LAPLACIAN KERNEL *:THE PROPOSED METHOD.

can result in higher accuracies. Finally we might investigate the adoption of this deep representation in other structured domains such as trees instead of graphs.

REFERENCES

[1] B. Schölkopf and A. J. Smola, *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[2] V. N. Vapnik, *Statistical learning theory*. Wiley New York, 1998.

[3] T. Hofmann, B. Scholkopf, and A. J. Smola, "Kernel methods in machine learning," *Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.

[4] C. Cortes and C. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[5] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International Workshop on Ambient Assisted Living*, 2012.

[6] D. Anguita, A. Ghio, L. Oneto, and S. Ridella, "In-sample model selection for trimmed hinge loss support vector machine," *Neural processing letters*, vol. 36, no. 3, pp. 275–283, 2012.

[7] M. Fernández-Delgado, E. Cernadas, S. Barro, and C. Amorim, "Do we need hundreds of classifiers to solve real world classification problems," *Journal Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[8] M. Wainberg, B. Alipanahi, and B. J. Frey, "Are random forests truly the best classifiers?" *Journal of Machine Learning Research*, vol. 17, no. 110, pp. 1–5, 2016.

[9] J. Tang, C. Deng, and G. B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 4, pp. 809–821, 2016.

[10] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[11] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[12] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[13] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[14] F. Fouss, K. Francoisse, L. Yen, A. Pirotte, and M. Saerens, "An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification," *Neural networks*, vol. 31, pp. 53–72, 2012.

[15] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[16] A. Reka and A. L. Barabasi, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, pp. 47–97, 2002.

[17] G. Bakir, T. Hofman, B. Schölkopf, A. J. Smola, B. Taskar, and S. V. N. Vishwanathan, *Predicting structured data*. MIT press, 2007.

[18] G. H. BakIr, J. Weston, and B. Schölkopf, "Learning to Find Pre-Images," in *Advances in neural information processing systems*, 2004.

[19] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, pp. 1157–1182, 2003.

[20] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[21] Y. Kim, H. Lee, and E. M. Provost, "Deep learning for robust feature generation in audiovisual emotion recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 3687–3691.

[22] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong, "Representational learning with elms for big data," *IEEE Intelligent Systems*, vol. 28, no. 6, pp. 31–34, 2013.

[23] P. Yanardag and S. V. N. Vishwanathan, "Deep graph kernels," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[24] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *International conference on Knowledge discovery and data mining*, 2014.

[25] A. Tikhonov and V. Y. Arsenin, *Methods for solving ill-posed problems*. Nauka, Moscow, 1979.

[26] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *International Conference on Computational Learning Theory*. Springer, 2001.

[27] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing multiclass to binary: A unifying approach for margin classifiers," *Journal of machine learning research*, vol. 1, pp. 113–141, 2000.

[28] J. Fürnkranz, "Round robin classification," *Journal of Machine Learning Research*, vol. 2, pp. 721–747, 2002.

[29] C. W. Hsu and C. J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[30] D. Anguita, A. Ghio, L. Oneto, and S. Ridella, "In-sample and out-of-sample model selection and error estimation for support vector machines," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 9, pp. 1390–1406, 2012.

[31] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence*, 1995.

[32] R. I. Lafferty and J. Kondor, "Diffusion kernels on graphs and other discrete structures," in *International Conference Machine Learning*, 2002.

[33] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Conference on learning theory*, 2003.

[34] F. Spitzer, "Reversibility and Stochastic Networks," *SIAM Review*, vol. 23, no. 3, pp. 400–401, 1981.

[35] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 191–218, 2006.

[36] B. L. Chen, M. Li, J. X. Wang, and F. X. Wu, "Disease gene identification by using graph kernels and Markov random fields," *Science China Life Sciences*, vol. 57, no. 11, pp. 1054–1063, 2014.

[37] A. Mantrach, N. Van Zeebroeck, P. Francq, M. Shimbo, H. Bersini, and M. Saerens, "Semi-supervised classification and betweenness computation on large, sparse, directed graphs," *Pattern Recognition*, vol. 44, no. 6, pp. 1212–1224, 2011.

[38] T. J. Abrahamsen and L. K. Hansen, "Regularized Pre-image estimation for kernel PCA de-noising : Input space regularization and sparse reconstruction," *Journal of Signal Processing Systems*, vol. 65, no. 3, pp. 403–412, 2011.

[39] NetKit-SRL, "Network Learning Toolkit for Statistical Relational Learning," http://netkit-srl.sourceforge.net/data.html, [Online; accessed 1-December-2016].

[40] WebKB Project, "CMU World Wide Knowledge Base (WebKB) project," http://www.cs.cmu.edu/ webkb/, [Online; accessed 1-December-2016].